Q.VITEC

Vision Q.400

Image Processing

Version 7.1.0.0 Q. HALCON Interface – Q.HI

Q.HI

HALCON-Interface

Q.400

Q.VITEC GmbH

Hagenburger Str. 54, 31515 Wunstorf, Germany

Tel.:+49(0)5031-949 43 20 Fax: +49(0)5031-949 43 29

Internet: www.qvitec.de

Date: 2014-09-04

Table of Contents

1 Introduction

Vision Q.400 utilizes MVTec's HALCON software for its machine vision algorithms. Sophisticated use of the HALCON **C/C++** library is behind each checker in Vision Q.400. However, as well as **C/C++**, HALCON offers a script based language for rapid development of machine vision algorithms and solutions. Scripts are written with the program HDevelop. These scripts can then be read by Vision Q.400 to provide user defined checkers.

When a script has been successfully loaded by Vision Q.400, it appears as an additional checker in the checkers toolbar.
This document describes the HDevelop script procedures required by Vision Q.400.

Software requirements

Other than Vision Q.400, the HALCON Development environment is required to create a script based checker. The HALCON version for developing the script can differ from Vision Q.400's HALCON version. However, you need to ensure that only procedures supported by the Vision Q.400's HALCON version are used. Scripts can then be developed with HDevelop and immediately used in Vision Q.400.

Minimum requirements for Vision Q.400 to load a HDevelop script based checker

To create a script based checker in Vision Q.400, there is surprisingly very little that needs be done. A HDevelop script just needs to be placed in the **.\User Defined Checkers\HDevelop** sub-folder of the Vision Q.400 installation. When Vision Q.400 starts, it will load all valid scripts that it finds (sub-folders are currently not searched) and make them available on the checkers toolbar.
A big advantage of scripting is that all procedures and their parameters are optional, so you only need to define the procedures and parameters that will be used. Any procedures or parameters that are not used can be entirely omitted. If the script is completely empty, default checker properties will be used. Of course such a script will not be very useful, but it could be the basis for creating a functional checker. Alternatively, one of the supplied example checkers could be used as a template for a new checker.

Example Scripts

A collection of sample scripts are supplied. For the Demo version they are immediately available, however for the Full version they need to be renamed in order to be loaded by Vision Q.400. Locate the scripts in the **.\User Defined Checkers\HDevelop** folder and rename them by removing the trailing underscore character.

Script Errors

If a script causes an error during loading, the Error Logging window should provide details of the error. This window is available from the toolbar or menu item View-Errors. Script runtime exception errors are displayed in a Vision Q.400 error dialog as well as the error log window.

Transfer Object, Transfer Tuple

Some procedures have as parameters a so called "transfer object" and a so called "transfer tuple". Both are not processed by Vision Q.400, and intended to be used only inside the HALCON script. They give the possibility to create them in one procedure and to use them in another, without implementing this "transfer" inside the HALCON script.

External Script Procedures

All external procedures located in the **.\User Defined Checkers** folder, including any sub-folders, are loaded when Vision Q.400 starts. If a new external procedure for a script is created, Vision Q.400 will need restarting in order to load the new procedure.

User Defined Checker DLLs

Vision Q.400's script based checkers are based on the "user defined checker" DLL technology. Therefore, there are many similarities between the two. Also, since HDevelop scripts can be exported to C++, converting a script based checker to a DLL based checker is relatively easy.

User defined checker DLL's have the filename extension **.uds** and are placed in the **.\User Defined Checkers\Uds** sub-folder of the Vision Q.400 installation.

2 HDevelop Procedures

Vision Q.400 uses script procedures with defined names and parameters. All defined procedures and their parameters are optional. If a procedure contains extra non Vision Q.400 parameters, then these will be ignored.

Essentially the procedures can be split into two groups. The first group is related to loading and giving the identity of the user defined checker. The second group is related to usage of the checker in an application.

Checker load and identity procedures

This group of procedures gives a checker its identity and determines its appearance in Vision Q.400. They are called once only when Vision Q.400 starts up and loads the checker.

• OnLoad	First procedure to be called when loading the checker.
• GetInfo	Determines checker identity.
• GetDescription	Describes the checker's general characteristics.
• GetParameterDescriptions	Describes parameters used by the checker.
• GetResultDescriptions	Describes results provided by the checker.
• GetUsedResultDescriptions	Describes the results that can be used from a previous checker.
• OnOpenApplication	Is called when an application is opened.
• OnCloseApplication	Is called when an application is closed.

If you modify any of these procedures, except for OnOpenApplication and OnCloseApplication, you need to restart Vision Q.400 in order for the changes to be realized.

Checker instance procedures

The instance group of procedures relate to each usage of the checker in the application. In other words, you create a checker instance when you insert a checker into the application, and thereby add it to the sequence list. Instance type procedures are called, when appropriate, for each checker instance in the sequence list.

• OnLoadInstance	A checker instance is loaded (Vision Q.400 loads the application).
• OnSaveInstance	A checker instance is saved (Vision Q.400 saves the application).
• OnCreateInstance	A checker instance is created (user inserts a checker).
• OnCopyInstance	A checker instance is copied (user copies a checker).
• OnDeleteInstance	A checker instance is deleted (user deletes a checker).
• OnExecuteInstance	A checker instance is executed.
• OnButtonPressed	A specific dialog button for the checker instance is pressed.

If you modify any of these procedures, no restart of Vision Q.400 is required in order to realize the changes.

2.1 OnLoad

Name

OnLoad — First procedure that is called when loading the checker. Called once only at startup.

Synopsis

OnLoad(: : LanguageId: LoadResult)

Description

Used to give the script an opportunity to decide whether the checker is to be loaded or not. For instance, this may be due to a language mismatch between Vision Q.400 and the script. If the checker is not loaded, it will not appear as an icon or menu entry in Vision Q.400. However, there will be an information entry added to the "Error Logging" window.

Parameters

LanguageId (input_control) **integer → (*integer*)**

Language that Vision Q.400 is running as.
The following table lists the supported values.

LanguageId Value	Meaning
1031	German
1033	English (United States)
1036	French

LoadResult (output_control) **integer → (*integer*)**

Result of the OnLoad procedure. Possible values are:

0 => The checker will not be loaded.
> 0 The checker will be loaded (Default)

Example

```
* For readability, define the possible LanguageId’s.
LANGUAGE_GERMAN := 1031
LANGUAGE_ENGLISH := 1033
LANGUAGE_FRENCH := 1036

* Script is only for English use.
if(LanguageId = LANGUAGE_ENGLISH)
    LoadResult := 1
else
    LoadResult := 0
endif
return ()
```

2.2 GetInfo

Name

GetInfo — called to get initial checker information. Called once only at startup.

Synopsis

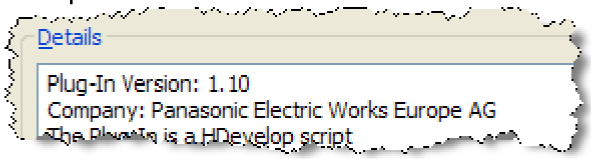
GetInfo(: : : Info)

Description

The returned information determines the checker naming, licensing and required HALCON version.
Key value pairs are added to the output tuple. e.g. `Info := [Info,0,'Example Company']` appends the key `0` and value `'Example Company'` to the output tuple Info.
For clarity, we assign the key values to tuples with an understandable name.

```
INFO_COMPANY_NAME := 0
INFO_HALCON_VERSION := 1
INFO_HALCON_REVISION := 2
INFO_SCRIPT_LICENSE := 3
```

The following table describes the key value pairs.

Key	Default Value	Values and Examples
`INFO_COMPANY_NAME`	'Example Company'	Any alphanumeric name. e.g. 'Company XYZ'. The company name is displayed in the dialog under the menu point Help/About Vision Q.400 Plug-Ins. For example:  Details Plug-In Version: 1.10 Company: Panasonic Electric Works Europe AG The PlugIn is a HDevelop script INFO_COMPANY_NAME and the parameter DESCRIPTION_NAME of the procedure GetDescription() are internally used to identify an user defined checker in a stored application. Therefore if either of these values are changed, an application that was saved with the old names cannot be read. So it is strictly recommended not to change INFO_COMPANY_NAME or DESCRIPTION_NAME.
`INFO_HALCON_VERSION`	HALCON Version of running Vision Q.400. This means that by default an attempt to load the script will be made by all versions of Vision Q.400.	Only set this key if you wish to restrict the script to running with a minimum HALCON version. For instance, the script uses functionality that is not available in lower versions of HALCON. Such functionality may lead to unexpected results or fatal errors in lower HALCON versions. e.g. setting to '10' will ensure that the script only runs with Vision Q.400 Version 6.0 and newer (HALCON version 10.0 is used starting with this version).
`INFO_HALCON_REVISION`	HALCON Revision of running Vision Q.400. This means that by default an attempt to load the script will be made by all versions of Vision Q.400.	Only use if you need to specify a minimum revision of the specified version to be required for the script. For instance, the script uses functionality that is not available in lower revisions of HALCON. Such functionality may lead to unexpected results or fatal errors in lower HALCON revisions. e.g. setting to '1', with version '10' would mean Vision Q.400 with HALCON 10 revision 1 as a minimum.

		Value	**Meaning**
`INFO_SCRIPT_LICENSE`	No license is required.	1(or do not provide the parameter at all, it has the same effect)	Licensed. Ensure that the procedure is password protected.
		0	Not licensed. The script will not be loaded.
		'SCRIPT_PEW'	A valid license for the string will be checked on the dongle.

Parameters

Info (output_control) **any → (*tuple*)**

Example

```
* Identifiers (The values should not be changed)
INFO_COMPANY_NAME := 0
INFO_HALCON_VERSION := 10
INFO_HALCON_REVISION := 2
INFO_SCRIPT_LICENSE := 3
Info := [INFO_COMPANY_NAME,'Example Company']
* HALCON version information. Do not use get_system for this. If required, hard code the minimum version
* that the script is meant to run under.
Info := [Info,INFO_HALCON_VERSION,'10.0']
* HALCON revision. Use in conjuction with the version if you want to specify a minimum revision.
Info := [Info,INFO_HALCON_REVISION,'0']
```

2.3 GetDescription

Name

GetDescription — called to obtain a list of the checkers features. Called once only at startup.

Synopsis

GetDescription(: : : Description)

Description

A procedure to get the Description is called once upon loading the checker when Vision Q.400 starts. The returned information specifies various properties of the checker. For instance, the checker's name and icon in the checker toolbar, which shapes it supports and whether it wants to use any of the inbuilt filtering, exposure adjustment or thresholding.

Parameters

Description (output_control) **any → (integer / string)**

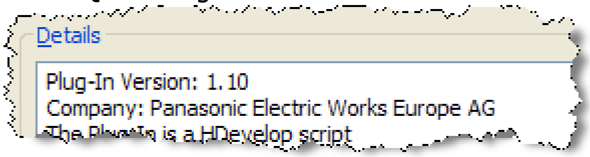
Key/Value pairs or entries are added to the output tuple.
e.g

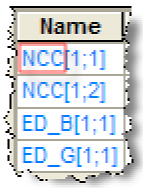
```
 Description := [Description, 0, ‘Checker ABC’]
```

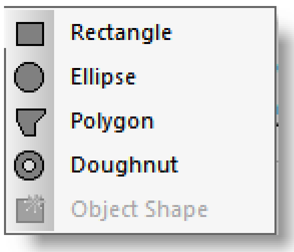
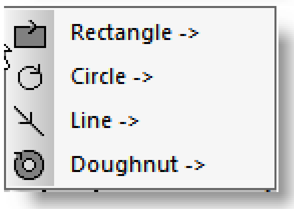
This appends the key `0` and value `'Checker ABC'` to the output tuple `Description`.

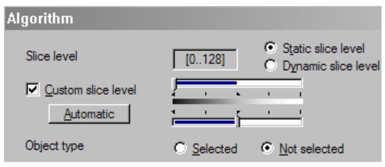
For clarity in a script, we assign the key values to tuples with an understandable name.

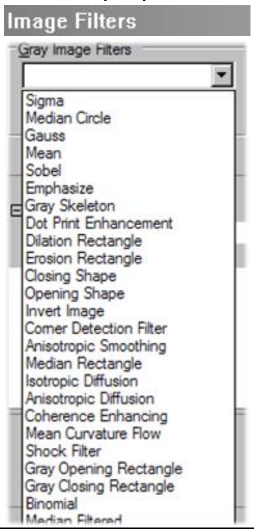
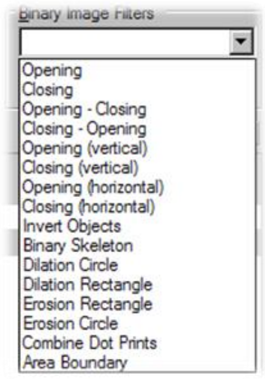
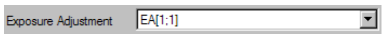
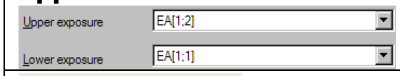
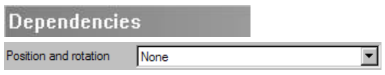
```
DESCRIPTION_VERSION := 0
DESCRIPTION_NAME := 1
DESCRIPTION_SHORT_NAME := 2
DESCRIPTION_SHAPES := 3
DESCRIPTION_COLORS := 4
DESCRIPTION_GRAY_FILTERS := 5
DESCRIPTION_BINARY_FILTERS := 6
DESCRIPTION_EXPOSURE_ADJUSTMENT := 7
DESCRIPTION_POS_ROT := 8
DESCRIPTION_BINARIZATION_PAGES := 9
DESCRIPTION_MAX_OBJECTS := 10
DESCRIPTION_CHECKER_ICON := 11
DESCRIPTION_PARAMETER_BUTTON_TIP := 12
DESCRIPTION_DEPENDENCIES_BUTTON_TIP := 13
```

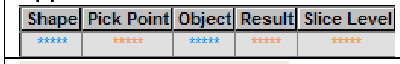
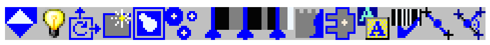
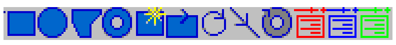
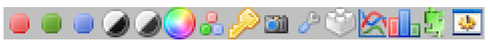
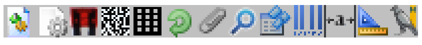
Key	Default	Values and Examples
DESCRIPTION_VERSION *Version number of the user defined checker.*	1	Any integer > 0 The least significant 4 bytes of the integer are used for the version. This gives the possibility to specify up to a four-digit version number. Generally a maximum of two digits will be required. The version is best defined by entering a hexadecimal value. For example: **Value** — **Represents version no.** 0x10000000 — 1.0.0.0 0x1001 — 1.1 0x010a — 1.10 The version of each plug-in script is displayed by Vision Q.400 in the dialog under the menu point Help/About Vision Q.400 Plug-Ins.  Details Plug-In Version: 1.10 Company: Panasonic Electric Works Europe AG The Plug-In is a HDevelop script

Key	Default	Values and Examples
DESCRIPTION_NAME Name of the checker. Name that appears in the Checker menu or toolbar of Vision Q.400.	Script file name, without an extension.	Any alphanumeric names. e.g. "Checker ABC". DESCRIPTION_NAME and the parameter INFO_COMPANY_NAME of the procedure GetInfo() are internally used to identify an user defined checker in a stored application. This means that if either DESCRIPTION_NAME or INFO_COMPANY_NAME are changed, an application that uses the old names cannot be read. Therefore it is strictly recommended not to change DESCRIPTION_NAME or INFO_COMPANY_NAME.
DESCRIPTION_SHORT_NAME Short name of the checker. 	Upper case of script file name without an extension.	Any alphanumeric name. Space characters are not allowed. First character cannot be numeric. For example, checker with short name NCC is shown. A suffix in the following format is automatically appended to the short name when a checker is added to the sequence list of a camera. **[<Camera No.>;<Checker No.>]** *where <Camera No.> is the camera to which the checker belongs and <Checker No.> is the number of that particular checker's instance for the specified camera.* This name, including suffix, also serves as the default ActiveX name.

DESCRIPTION_SHAPES
Possible shapes allowed for defining the checkers shape region.


	rectangle
	ellipse
	polygon
	doughnut
	object
		Comma separated list of any of the following:
		`rectangle` `rectangle ->`
		`ellipse` `ellipse ->`
		`doughnut` `doughnut ->`
		`polygon` `line ->`
		`object shape`
		NOTE: there needs to be a space character before the `->`
		
		e.g.
		`Description := [Description,DESCRIPTION_SHAPES,'rectangle, ellipse, doughnut, line ->']`
		Additionally, a blank string can be defined. e.g.
		`Description := [Description,DESCRIPTION_SHAPES,'']`
		This would result in a checker with no shape.

DESCRIPTION_BINARIZATION_PAGES Availability of the Algorithm, Color Selection and Channel Selection tab in the checker properties dialog. This provides Vision Q.400's default binarization controls.  Algorithm Slice level [0..128] Static slice level Dynamic slice level Custom slice level Automatic Object type Selected Not selected	0	0 => No tab for binarization is shown. 1 => Algorithm tab. When the checker works on a color image, the color selection and channel selection tab are included. 3 => Algorithm tab with "Object type" option. When the checker works on a color image, the color selection and channel selection tab are included. 4 => Only the "Color Selection" tab is available. 8 => Only the "Channel Selection" tab is available This has to be set to OFF if the checker has no shapes (see *DESCRIPTION_SHAPES*)

Key	Default	Values and Examples
DESCRIPTION_GRAY_FILTERS Availability of Gray filters on the Image Filters tab of the checker properties dialog.  Image Filters Gray Image Filters Sigma Median Circle Gauss Mean Sobel Emphasize Gray Skeleton Dot Print Enhancement Dilation Rectangle Erosion Rectangle Closing Shape Opening Shape Invert Image Corner Detection Filter Anisotropic Smoothing Median Rectangle Isotropic Diffusion Anisotropic Diffusion Coherence Enhancing Mean Curvature Flow Shock Filter Gray Opening Rectangle Gray Closing Rectangle Binomial Median Filtered	0	0 => OFF 1 => ON This has to be set to OFF if the checker has no shapes (see *DESCRIPTION_SHAPES*)
DESCRIPTION_BINARY_FILTERS Availability of Binary filters on the Image Filters tab of the checker properties dialog.  Binary Image Filters Opening Closing Opening - Closing Closing - Opening Opening (vertical) Closing (vertical) Opening (horizontal) Closing (horizontal) Invert Objects Binary Skeleton Dilation Circle Dilation Rectangle Erosion Rectangle Erosion Circle Combine Dot Prints Area Boundary	1	0 => OFF 1 => ON This has to be set to OFF if the checker has no shapes (see *DESCRIPTION_SHAPES*)
DESCRIPTION_EXPOSURE_ADJUSTMENT Availability of Exposure Adjustment on the Dependencies tab of the checker properties dialog. **ON** Exposure Adjustment EA[1;1] or **ON distinguishing between upper and lower** Upper exposure EA[1;2] Lower exposure EA[1;1]	2	0 => OFF 1 => ON 2 => ON distinguishing between Upper and Lower This has to be set to OFF if the checker has no shapes (see *DESCRIPTION_SHAPES*)
DESCRIPTION_POS_ROT The checker can use a position rotation adjustment.  Position and rotation None	1	0 => Not available 1 => Available This has to be set to 0 (Not available) if the checker has no shapes (see *DESCRIPTION_SHAPES*)

Key	Default	Values and Examples
DESCRIPTION_COLORS Specifies which colors of the checker are **allowed** to be modified in the dialog under the Vision Q.400 menu Application -> Colors 	All colors for the checker can be changed.	shape pick point object result slice level e.g. `'shape, result'`
DESCRIPTION_MAX_OBJECTS Maximum number of detected objects for which results will be calculated.	128	This must be bigger than zero.
DESCRIPTION_CHECKER_ICON The icon given to the checker in the checkers toolbar and beside it's entry in the menu.	26	Number of one of the provided icons OR any bitmap (BMP) filename. For a bitmap, it should be at least 16x16 pixels in size. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  16 17 18 19 20 21 22 23 24 25 26 27  28 29 30 31 32 33 34 35 36 37 38 39 40 41 42  43 44 45 46 47 48 49 50 51 52 53 54 55  e.g. `'35'` e.g. `'Checker.bmp'` The bitmap file should be located in the same folder as the script file.
DESCRIPTION_PARAMETER_BUTTON_TIP 	No tooltip	Tooltip belonging to the button on the Parameters page
DESCRIPTION_DEPENDENCIES_BUTTON_TIP 	No tooltip	Tooltip belonging to the button on the Dependencies page

Example

```
* Version of the script (0x0010a => 1.10)
Description := [DESCRIPTION_VERSION,0x0010a]
* The following two values need to be unique for each script
Description := [Description,DESCRIPTION_NAME,'Example Script 1']
Description := [Description,DESCRIPTION_SHORT_NAME,'HS_EX1']
* Shapes available for checker - default is "rectangle, ellipse, polygon, object"
Description := [Description,DESCRIPTION_SHAPES,'rectangle, ellipse, doughnut']
* Specify the checker objects types that can have their color changed. Default: all checker object types.
Description := [Description,DESCRIPTION_COLORS,'shape']
* Gray filters setting - default is OFF
Description := [Description,DESCRIPTION_GRAY_FILTERS,1]
* Binary filters setting - default is ON
Description := [Description,DESCRIPTION_BINARY_FILTERS,1]
Description := [Description,DESCRIPTION_EXPOSURE_ADJUSTMENT,1]
Description := [Description,DESCRIPTION_POS_ROT,0]
Description := [Description,DESCRIPTION_BINARIZATION_PAGES,3]
Description := [Description,DESCRIPTION_MAX_OBJECTS,128]
```

```
* Icon can be either a bitmap file name (must be in same directory as script) or an index of a standard
supplied bitmap. Indexes 0-31 are valid.
Description := [Description,DESCRIPTION_CHECKER_ICON,'31']
Description := [Description,DESCRIPTION_PARAMETER_BUTTON_TIP,'Click here to execute OnButtonPressed']
Description := [Description,DESCRIPTION_DEPENDENCIES_BUTTON_TIP,'Click here to execute OnButtonPressed']
```

2.4 GetParameterDescriptions

Name

GetParameterDescriptions — called to obtain a list of parameters that the checker wishes to show in the user interface. Called once only at startup.

Synopsis

GetParameterDescriptions(: : : ParameterDescriptions, SlaveDescriptions, AcessMode, Properties)

Description

Used for specifying user modifiable parameters required by the checker. Parameters can be conditionally displayed.

Parameter

ParameterDescriptions (output_control) **any → (integer / real / string)**

Tuple containing entries for each parameter that the checker needs to expose to the user. For each parameter type, the number of entries in the tuple may differ.

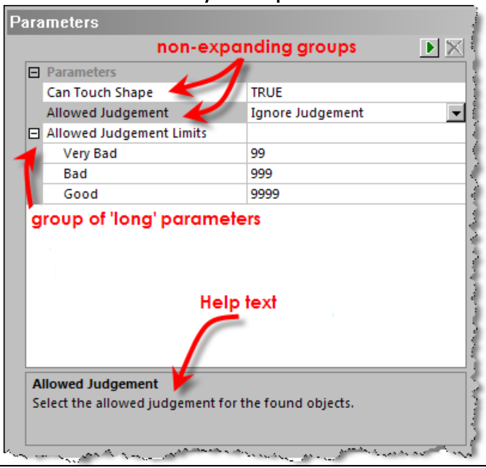
NOTE: Parameter type names are case sensitive.

NOTE: In the following description the index for each parameter type starts with 0, which is not correct in a real script, since the descriptions of all parameters are contained in one tuple.

NOTE: If the tuple does not start with a "Group" parameter, a "Group" parameter called "Group 1" with no help text is inserted at the head of the tuple.

Group

- Requires 3 tuple entries
- May not be empty. In other words, it must be followed by one or more parameter descriptions.
- If a Group parameter is not defined, a group named "Parameters" with no help text is used.
- If the name of the group is the same as the name of the first parameter it contains, then the group item will not be expandable in the list (see diagram below). This feature is especially useful for "Group" parameters which contain only one parameter.



⊟ Parameters	
Can Touch Shape	TRUE
Allowed Judgement	Ignore Judgement
⊟ Allowed Judgement Limits	
Very Bad	99
Bad	999
Good	9999

Allowed Judgement
Select the allowed judgement for the found objects.

Entry No.	Type	Description
0	string	`'Group'`
1	string	Name of the parameter group.
2	string	A help text displayed for the parameter group. This text may be empty.

Non-Group Parameters.

The tuple should contain at least one non-group parameter.

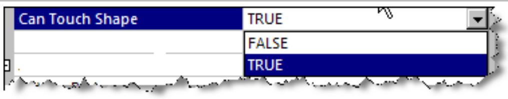
The following non-group parameters can be defined.

Bool	Long	Enum
Double	String	

Bool

- Requires 4 tuple entries.
- e.g.

```
ParameterDescriptions := ['Group','Can Touch Shape','Select "TRUE" if it is allowed that an object touches the checker shape, "FALSE" otherwise.']
ParameterDescriptions := [ParameterDescriptions,'Bool','Can Touch Shape','true','Select "TRUE" if it is allowed that an object touches the checker shape, "FALSE" otherwise.']
```

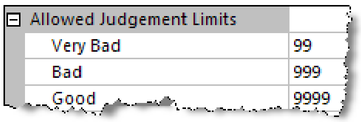


Entry No.	Type	Description
0	string	'Bool'
1	string	Name of the parameter.
2	string	Default value of the parameter: "true" or "false".
3	string	A help text displayed for the parameter. This text may be empty.

Long

- Requires 7 tuple entries.
- e.g.

```
LONG_MIN := -0x7fffffff-1
LONG_MAX := 0x7fffffff
ParameterDescriptions := [ParameterDescriptions,'Group','Allowed Judgement Limits','The limits of the different judgements can be adjusted here.']
ParameterDescriptions := [ParameterDescriptions,'Long','Very Bad',99,LONG_MIN,LONG_MAX,'','Upper limit of judgement "Very Bad".']
ParameterDescriptions := [ParameterDescriptions,'Long','Bad',999,LONG_MIN,LONG_MAX,'GE','Upper limit of judgement "Bad".']
ParameterDescriptions := [ParameterDescriptions,'Long','Good',9999,LONG_MIN,LONG_MAX,'GE','Upper limit of judgement "Good".']
```



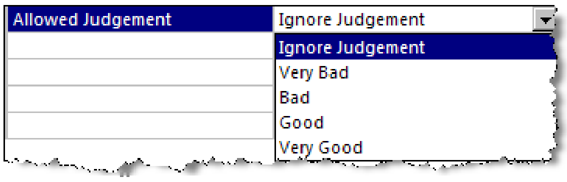
Allowed Judgement Limits	
Very Bad	99
Bad	999
Good	9999

Entry No.	Type	Description
0	string	'Long'
1	string	Name of the parameter.
2	integer	Default value of the parameter.
3	integer	Minimum value of the parameter.
4	integer	Maximum value of the parameter.
5	string	Dependency to the parameters predecessor. The predecessor has to be a "Long" parameter. Set to empty string, if a dependency does not exist. Set to "GT" if the parameter has to be bigger as it's predecessor in the list. Set to "GE" if the parameter has to be bigger as or equal to it's predecessor in the list.
6	string	A help text displayed for the parameter. This text may be empty.

Enum

- Described by a variable number of tuple entries.
- e.g.

```
ParameterDescriptions := [ParameterDescriptions,'Enum','Allowed Judgement',0,5,'Ignore Judgement', 'Very Bad', 'Bad',
'Good', 'Very Good','Select the allowed judgement for the found objects.']
```

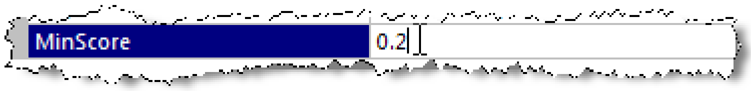


Entry No.	Type	Description
0	string	`'Enum'`
1	string	Name of the parameter.
2	integer	Default index into the enumeration array. The index starts with zero.
3	integer	Number of enumeration values. This number is limited to 15.
4 .. (4+Entry 3 – 1)	string	Name of the enumeration value.
4 + Entry 3	string	A help text displayed for the parameter. This text may be empty.

Double

- Requires seven tuple entries.
- e.g.

```
ParameterDescriptions := [ParameterDescriptions,'Double','MinScore',0.1,0.1,1.0,'','Minimum score']
```

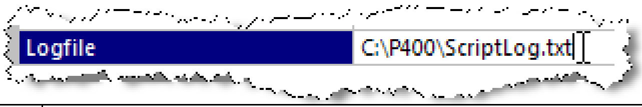


Entry No.	Type	Description
0	string	`'Double'`
1	string	Name of the parameter.
2	real	Default value of the parameter.
3	real	Minimum value of the parameter.
4	real	Maximum value of the parameter.
5	string	Dependency to the parameters predecessor. The predecessor has to be of a "Double" parameter. Set to empty string, if a dependency does not exist. Set to "GT" if the parameter has to be bigger than it's predecessor in the list. Set to "GE" if the parameter has to be bigger as or equal to it's predecessor in the list.
6	string	A help text displayed for the parameter. This text may be empty.

String

- Requires 4 entries.
- e.g.

```
ParameterDescriptions := [ParameterDescriptions,'String','Logfile','C:\\P400\\ScriptLog.txt','Specifies logfile']
```



Entry No.	Type	Description
0	string	`'String'`
1	string	Name of the parameter.
2	string	Default value of the parameter.
3	string	A help text displayed for the parameter. This text may be empty.

Changing a parameter name
Parameter names are saved together with their value, so if a name is changed in the script it will not receive the correct value when applications saved from old scripts are loaded. In this case the old name should be kept in the parameter descriptions list and be set as hidden via the **Fehler! Verweisquelle konnte nicht gefunden werden.** parameter (see below). The script can then deal with the old parameter if it was used.

Different languages
As mentioned above, parameter names are stored in the application. This applies to group names too. Therefore a problem is likely to occur if the script language changes. For example, an application saved with an English script is read using a German script. Since the English parameter names are different, the values cannot be assigned when the application is read. Unassigned parameters will receive their default value.

To solve this problem, every parameter, group, and enumeration value can be assigned a language independent name and a localized name. Note that if a parameter receives a language independent name, its group needs one as well.
When a language independent name is used, it is saved with the application and should also be used for access via ActiveX. The localized name can then be safely modified without causing an issue in the application or ActiveX client.

To specify two names, the language independent name should follow the localized name. They should be separated by the "@A@" character sequence.
For example, "Steigende Flanke@A@Rising Edge" would result in a localized name "Steigende Flanke" and a language independent name "Rising Edge".

SlaveDescriptions (output_control) **any → (integer / string)**
This tuple describes the visibility dependencies between parameters. A visibility dependencies means that the visibility of a parameter (or a parameter group) depends on the current value of another parameter. (For an example see the indicators in Vision Q.400.)
A parameter that controls the visibility of another parameter (or parameter group), is called the master, and the dependent parameter (or parameter group) is called the slave. Only Bool and Enum type parameters can be master.
For every visibility dependency, the number of entries in the tuple may differ. In the following description, the index for each visibility dependency starts with 0
Each parameter or group is defined by 4 tuple entries as follows:

Entry No.	Type	Description
0	string	Master's group name
1	string	Master's parameter name
2	string	Slave's group name
3	string	Slave's parameter name. If this is an empty string, the slave is the whole group, as given by the name in entry 2.
4	integer	Number of values of the master, for which the slave is visible. This number is limited to 15. If this entry is 0, the slave parameter or parameter group is always hidden. The master is not considered in this case.
5..(5+Entry 4 – 1) e.g. Entry 4 is 1 => 5..5 Entry 4 is 2 => 5..6	string	Name of the enumeration value of the master, for which the slave is visible. For all other values of the master the slave is invisible.

AccessMode (output_control) **any → (integer / string)**

Tuple which contains additional info about the access mode of parameters. The access mode info allows definition of parameters or groups of parameters that are to be read only or hidden in the user interface. The parameters are nevertheless saved in the application.

The access mode of a parameter or group is defined by for tuple entries as follows:

Entry No.	Type	Description
0	string	The name of the group.
1	string	The name of the parameter (or empty if the access mode belongs to a whole group). NOTE: If the group has only one parameter and both the group and parameter names are the same, set this empty.
2	integer	Read only flag. Set to 1 if the parameter or group is read only, otherwise 0.
3	integer	Hidden flag. Set to 1 if the parameter or group is hidden, otherwise 0.

Properties (output_control) **any → (integer / string)**

Tuple which contains additional info about the properties of parameters.
The following properties are available:

Property	Allowed for Parameter Type	Description
<CheckBox>	Bool	In the GUI of Vision Q.400 a check box is displayed to select / deselect the value.
<Color>	Long	The long parameter is a color value. In the GUI of Vision Q.400 an additional color selection box is displayed to change the parameter's color value.
<SpinButton>	Long	In the GUI of Vision Q.400 an additional spin button is displayed to change the parameter's value.
<File>	String	The string parameter represents a file name. In the GUI of Vision Q.400 an additional file selection dialog is displayed to change the parameter's value. The property <File> has three additional parameters: OpenOrSave: Open: an "open" file dialog is displayed. Save: a "save" file dialog is displayed. DefaultExtention: The default extension for the file dialog, e.g. "*.txt". FileFilter: The file filter for the file dialog, e.g. "Text Files (*.txt)|*.txt||". The following example opens text files: <File>Open,*.txt, Text Files (*.txt)|*.txt||

<Directory>	String	The string parameter represents a directory name. In the GUI of Vision Q.400 an additional directory selection dialog is displayed to change the parameter's value. The property <Directory> has one additional parameter: OpenOrSave: Open: an "open" directory dialog is displayed. Save: a "save" directory dialog is displayed. The following example opens an existing directory: <Directory>Open

A property of a parameter is defined by three tuple entries as follows:

Entry No.	Type	Description
0	string	The name of group.
1	string	The name of the parameter.
2	integer	The property of the parameter.

Example

```
ParameterDescriptions := ['Group','Can Touch Shape','Select "TRUE" if it is allowed that an object touches
the checker shape, "FALSE" otherwise.']
ParameterDescriptions := [ParameterDescriptions,'Bool','Can Touch Shape','true','Select "TRUE" if it is
allowed that an object touches the checker shape, "FALSE" otherwise.']
ParameterDescriptions := [ParameterDescriptions,'Group','Allowed Judgement','Select the allowed judgement
for the found objects.']
ParameterDescriptions := [ParameterDescriptions,'Enum','Allowed Judgement',0,5,'Ignore Judgement', 'Very
Bad', 'Bad', 'Good', 'Very Good','Select the allowed judgement for the found objects.']
ParameterDescriptions := [ParameterDescriptions,'Group','Allowed Judgement Limits','The limits of the
different judgements can be adjusted here.']
LONG_MIN := -0x7fffffff-1
LONG_MAX := 0x7fffffff
IGNORE_JUDGEMENT := 0
VERY_BAD := 1
BAD := 2
GOOD := 3
VERY_GOOD := 4

ParameterDescriptions := [ParameterDescriptions,'Long','Very Bad',99,LONG_MIN,LONG_MAX,'','Upper limit of
judgement "Very Bad".']
ParameterDescriptions := [ParameterDescriptions,'Long','Bad',999,LONG_MIN,LONG_MAX,'GE','Upper limit of
judgement "Bad".']
ParameterDescriptions := [ParameterDescriptions,'Long','Good',9999,LONG_MIN,LONG_MAX,'GE','Upper limit of
judgement "Good".']
ParameterDescriptions := [ParameterDescriptions,'Group','Logfile','Specifies the logfile that will be
created.']
ParameterDescriptions := [ParameterDescriptions,'String','Logfile','ScriptLog.txt','Specifies the logfile
that will be created.']
ParameterDescriptions := [ParameterDescriptions,'Group','MinScore','']
ParameterDescriptions := [ParameterDescriptions,'Double','MinScore',0.1,0.1,1.0,'','Minimum score of the
returned poses']

* Use SlaveDescriptions to define parameters that are conditionally displayed according to another
parameter’s setting.
* ‘Allowed Judgement Limits’ are only shown in ‘Allowed Judgement’ is NOT ‘Ignore Judgement’
SlaveDescriptions := ['Allowed Judgement','Allowed Judgement','Allowed Judgement Limits','',4,'Very
Bad','Bad','Good','Very Good']

* Use AccessMode to set parameters Hidden or Read Only.
AccessMode := []
* Note that since the group and parameter names are identical, and there is only one parameter, the
parameter name is blank.
* Make ‘Logfile’ Hidden
AccessMode:= [AccessMode,'Logfile','',0,1]
* Make ‘MinScore’ read only.
AccessMode:= [AccessMode,' MinScore ','',1,0]
```

2.5 GetResultDescriptions

Name

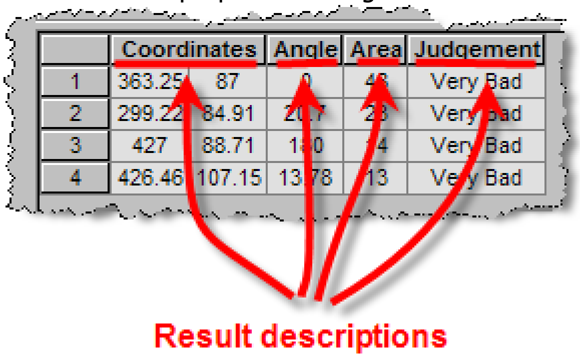
GetResultDescriptions — called to obtain the number and type of results supplied by the checker. Called once only at startup.

Synopsis

GetResultDescriptions(: : : ResultDescriptions)

Description

The returned information specifies the number and type of results supplied by the checker. The results are displayed on the Result tab of the checker's properties dialog.



	Coordinates		Angle	Area	Judgement
1	363.25	87	0	48	Very Bad
2	299.22	84.91	20.7	23	Very Bad
3	427	88.71	180	14	Very Bad
4	426.46	107.15	13.78	13	Very Bad

Parameters

ResultDescriptions (output_control) **any → (integer / string)**

Tuple containing 4 entries for each result that the checker wants to expose to the user on the Result tab. In other words each column of the table is defined by 4 entries in this tuple. The results in the table shown above are defined by 4 X 4 = 16 entries.

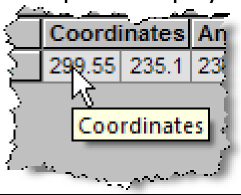
The names at the end of each description are used in error messages.

Entries for a "set" of results.		
Entry No.	**Type**	**Description**
0	string	Result Type The following values are supported.
		Result Type — **Description**
		'Double' — The result is one double value.
		'String' — The result is one string value.
		'Angle180' — The result is one double value, representing an angle in the range of 0 .. 180 degrees.
		'Angle360' — The result is one double value, representing an angle in the range of 0 .. 360 degrees.
		'Coordinate' — The results are two double values, the first represents the X coordinate, and the second the Y coordinate.
		'Rectangle' — The results are four double values, the first represents the X coordinate of the upper left corner, the second the Y coordinate of the upper left corner, the third the X coordinate of the bottom right corner, and the forth the Y coordinate of the bottom right corner.
		e.g. 'Coordinate'
1	string	Column header of the column in which the result is displayed. e.g. 'Coordinates'

2	string	Draw Hint The draw hint contains information about how a result should be drawn. The drawing order is as follows: 1. The objects given in the in OnExecuteInstance output parameter ObjectsWithResults. 2. The objects given in the in OnExecuteInstance output parameter ObjectsToDraw. 3. The results according to their draw hints. The following values are supported for the different Result Types:

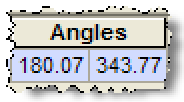
Result Type	Available draw Hints.	Description
Double	''	The result is not drawn.
String	''	The result is not drawn.
Angle180 Angle360	''	The result is not drawn.
	'arrow<position>'	Draws an arrow with the direction of the angle result at position <position>. <position> has to be the name of a "Coordinate" result, e.g. "arrow<Center>" draws the arrow at the position of the "Coordinate" result Center. If <position> is empty, that means only "arrow" is given, the position of the arrow has to be given by a "Coordinate" result which immediately precedes the angle result.
Coordinate	''	The result is not drawn.
	'cross'	Draws a cross at the position given by the "Coordinate" result.
	'circle<radius>'	Draws a circle with the radius <radius> at the position given by the "Coordinate" result. <radius> has to be the name of a "Double" result. E.g. "circle<Radius>" draws a circle with the radius given by the "Double" result Radius. If <radius> is empty, that means only "circle" is given, the radius of the circle has to be given by a "Double" result which immediately follows the "Coordinate" result.
	'circlecross<radius>'	Combination of "circle<radius>" and "cross". Means a circle with a center.
Rectangle	'rectangle'	Draws the rectangle given by the "Rectangle" result.

e.g. 'cross'

3	string	A help text displayed for the result as a tooltip. This text may be empty. 

If the column header names of consecutive result descriptions are the same, these results are displayed in separate columns, but they will share the same column header.
e.g.

```
ResultDescriptions := [ResultDescriptions, 'Angle180', 'Angles', '', '']
ResultDescriptions := [ResultDescriptions, 'Angle360', 'Angles', '', '']
```



The two values of a "Coordinate" result, and the four values of a "Rectangle" result, share the same column header, too.

Different languages
User defined checker results can be accessed by an ActiveX client. This is done with the help of the name of a result. Therefore an issue may occur if the language of a user defined checker dll is for example changed English to German, and an ActiveX client was written using the English names of results. Since the English names of results used in the ActiveX client do not fit the current German names of results, the results cannot be accessed by the client using the English names: these names are unknown to the user defined checker.

To solve this problem, every result can be assigned a so called language independent name in addition to the localized name. If such a language independent name is assigned, it is used to access the result values by an ActiveX client. That means as long as the "language independent" name is not changed, the language of the "normal" name can be changed, but the checker's results can be accessed by an ActiveX client without problems.

To assign a language independent name, it needs to be appended to the localized name. The two names should be separated by the "@A@" character sequence.
For example, "Fläche@A@Area" would result in a localized name "Fläche", but for the access by an ActiveX client "Area" should be used.

Example

```
ResultDescriptions := []
ResultDescriptions := ['Coordinate','Coordinates','cross','']
ResultDescriptions := [ResultDescriptions,'Angle360','Angle','arrow','']
ResultDescriptions := [ResultDescriptions,'Double','Area','','']
ResultDescriptions := [ResultDescriptions,'String','Judgement','','']
return ()
```

2.6 GetUsedResultDescriptions

Name

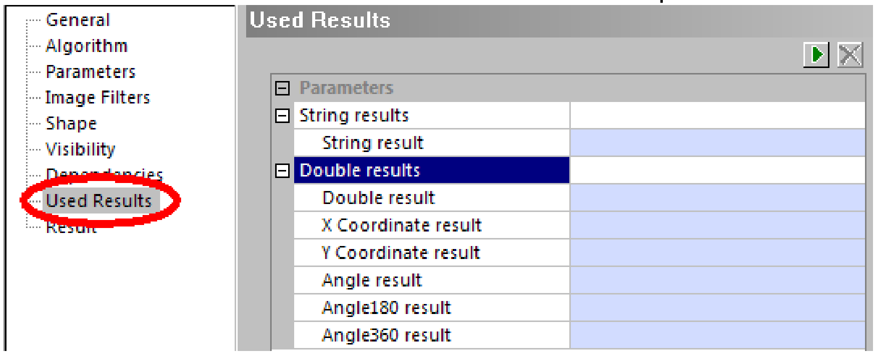
GetUsedResultDescriptions — called to obtain a list of the results that should be obtainable from a previous checker. Called once only at startup.

Synopsis

GetUsedResultDescriptions(: : : UsedResultDescriptions, WhenToCall)

Description

GetUsedResultDescriptions is for specifying results that the checker is able to receive from a previous checker in the sequence list. If this procedure is implemented, an extra tab named "Used Results" is added to the checker's property dialog. The "Used Results" property page contains a list of results that the checker can make use of in the OnExecute or OnButtonPressed procedures.



The Used Results are displayed in the parameters tables, as shown in the above example.

If the procedure is not defined, or the tuple UsedResultDescriptions is empty, the Used Results tab of the checker's property page will not be available.

Parameters

UsedResultDescriptions (output_control) **any → (string)**

Tuple containing string entries that define the table of Parameters on the Used Results tab of the checker's property dialog.
3 tuple entries define each row in the parameters table. The parameters are listed in groups in the table. To define a group the tuple receives a `Group` definition.

Entry No.	Supported values	Meaning in parameters table
0	'Group'	group
	'Double'	double value result can be dropped into table
	'String'	string value result can be dropped into table
	'Angle'	angle value result (any type) can be dropped into table
	'Angle180'	Inertia Axis (0-180) type angle result value can be dropped into table
	'Angle360'	Orientation (0-360) type angle result value can be dropped into table
	'Coordinate'	X or Y coordinate result value can be dropped into table
	'CoordinateX'	X coordinate result value can be dropped into table
	'CoordinateY'	Y coordinate result value can be dropped into table
1	Any text	Name in the first column of the parameters table.
2	Any text	Help text in the area below the parameters table.

WhenToCall (output_control) **integer → (integer)**

An Integer value, which defines how Vision Q.400 will behave if it should call a procedure which uses results, e.g. OnExecuteInstance(), and the execution result of at least one of the checkers, which calculate the results, is not OK.

0: The procedure will not be called. This is the default.

1: The procedure will be called if the execution results of the not OK checkers can be forced to NG.

2: The procedure will always be called. This means that the execution results of the not OK checkers are ignored.

Example

For example, in the 'Used Results' table that is shown above.

The first group called 'String results' is defined in a `Group` definition:

```
   UsedResultDescriptions := ['Group','String results', 'Demonstrates the use of a string result']
```

Following the group definition are the result entries, in this case just one being appended to the tuple:

```
  UsedResultDescriptions := [UsedResultDescriptions,'String','String result', 'Demonstrates the use of a
string result']
```

A new group definition then follows:

```
  UsedResultDescriptions := [UsedResultDescriptions,'Group','Double results', 'Demonstrates the use of
double results']
```

Followed by the result entries:

```
UsedResultDescriptions := [UsedResultDescriptions,'Double','Double result', 'Demonstrates the use of a
double result']
UsedResultDescriptions := [UsedResultDescriptions,'CoordinateX','X Coordinate result', 'Demonstrates the
use of an X coordinate result']
UsedResultDescriptions := [UsedResultDescriptions,'CoordinateY','Y Coordinate result', 'Demonstrates the
use of a Y coordinate result']
UsedResultDescriptions := [UsedResultDescriptions,'Angle','Angle result', 'Demonstrates the use of an
angle result']
UsedResultDescriptions := [UsedResultDescriptions,'Angle180','Angle180 result', 'Demonstrates the use of
an angle result']
UsedResultDescriptions := [UsedResultDescriptions,'Angle360','Angle360 result', 'Demonstrates the use of
an angle result']
```

2.7 OnOpenApplication

Name

OnOpenApplication — called when an application is opened.

Synopsis

OnOpenApplication(: : Start, Name, Result)

Description

OnOpenApplication is called for each script when an application is opened. The procedure is called even if the script is not used in the application. Note that this could unnecessarily reduce performance if scripts that use this functionality are never used.

Parameters

Start (input_control) **any → (integer)**

On opening an application the procedure is called twice:

1: the opening of the application is started.

0: the opening of the application is finished.

Name (input_control) **any → (string)**

The name of the application which is opened. If Name is empty, an application is not opened, but created.

Result (input_control) **any → (integer)**

The result of the opening:

1: the opening of the application was successful.

0: the opening of the application failed.

If bStartOpening is 1, bOpenResult is always 0.

2.8 OnSaveApplication

Name

OnSaveApplication — called when an application is saved.

Synopsis

OnSaveApplication(: : Start, Name)

Description

OnSaveApplication is called for each script when an application is saved, and can, for example, be used to prepare the application saving in the script. The procedure is called even if the script is not used in the application. Note that this could unnecessarily reduce performance if scripts that use this functionality are never used.

Parameters

Start (input_control)	**any → (integer)**

On saving an application the procedure is always called twice:

1: the saving of the application is started.

0: the saving of the application is finished.

Name (input_control)	**any → (string)**

The name of the application which is saved.

2.9 OnCloseApplication

Name

OnCloseApplication — called when an application is closed.

Synopsis

OnCloseApplication(: : Start, Name)

Description

OnCloseApplication is called for each script when an application is closed, and can, for example, be used to clear memory acquired by the script. The procedure is called even if the script is not used in the application. Note that this could unnecessarily reduce performance if scripts that use this functionality are never used.

Parameters

Start (input_control) **any → (integer)**

On closing an application the procedure is always called twice:

1: the closing of the application is started.

0: the closing of the application is finished.

Name (input_control) **any → (string)**

The name of the application which is closed. If Name is empty, an application was never saved.

2.10 OnLoadInstance

Name

OnLoadInstance — called when a checker instance has been loaded from file. This occurs when an application is opened.

Synopsis

OnLoadInstance(Object : ObjectToTransfer : Cookie, CheckerVersion, ParameterValues, TupleObject, Tuple : TupleToTransfer, NewCookie)

Description

OnLoadInstance is called directly after an instance of the user defined checker is read from a file.

Parameters

Object (input_object)	**object(-array)→ object**

The content of Object has been loaded by Vision Q.400 from the file. (See OnSaveInstance.)

ObjectToTransfer (output_object)	**object(-array) → object**

The transfer object of the checker instance. It is given here to give the checker instance the chance to initialize it with data stored in the input_object Object.

Cookie (input_control)	**any → (integer)**

Cookie which may help to identify a user defined checker instance. The cookie may especially be useful if the script manages different information for each instance.

CheckerVersion (input_control)	**any → (integer)**

The read in checker version. This is the value that was given by the GetDescription procedure.
(see *DESCRIPTION_VERSION*)

ParameterValues (input_control)	**any → (integer / real / string)**

Contains the current parameter values in the same order the parameters are given.

TupleObject (input_control)	**any → (integer / real / string)**

The content of TupleObject has been loaded by Vision Q.400 from the file. (See OnSaveInstance.)

Tuple (input_control)	**any → (integer / real / string)**

The content of Tuple has been loaded by Vision Q.400 from the file. (See OnSaveInstance.)

TupleToTransfer (output_control)	**any → (integer / real / string)**

The transfer tuple of the checker instance. It is given here to give the checker instance the chance to initialize it with data stored in the input_control ObjectTuple.

NewCookie (output_control)	**any → (integer)**

If set, this new Cookie value will be assigned to this checker instance.

2.11 OnSaveInstance

Name

OnSaveInstance — called when a checker instance is about to be saved to file. This occurs on application save.

Synopsis

OnSaveInstance(ObjectToTransfer : Object : Cookie, ParameterValues, TupleToTransfer : TupleObject, Tuple)

Description

OnSaveInstance is called directly before an instance of the user defined checker is stored to a file.

Parameters

ObjectToTransfer (input_object) **object(-array)→ object**
The transfer object of the checker instance. This is not stored automatically by Vision Q.400 to the file. Instead it is given here to give the checker instance the chance to insert it, or a part of it, into the output_object "Object", which is stored by Vision Q.400 to the file.

Object (output_object) **object(-array) → object**
The content of Object is stored by Vision Q.400 to the file. If an object to be stored is an image, the current domain of the image is stored, too.

Please be aware that if Object contains large objects, e.g. large images, the size of the application file may increase drastically. And this means that the loading time of the application file may increase drastically, too.

Cookie (input_control) **any → (integer)**
Cookie which may help to identify a user defined checker instance. The cookie may especially be useful if the script manages different information for each instance.

ParameterValues (input_control) **any → (integer / real / string)**
Contains the parameter values that will be stored for this checker instance.

TupleToTransfer(input_control) **any → (integer / real / string)**
The transfer tuple of the checker instance. This is not stored automatically by Vision Q.400 to the file. Instead it is given here to give the checker instance the chance to insert it, or a part of it, into the output_control "TupleObject", which is stored by Vision Q.400 to the file.

TupleObject (output_control) **any → (integer / real / string)**
The content of TupleObject is stored by Vision Q.400 to the file. It is intended to contain information about the output_object Output, but can be used in another way, too.

Tuple (output_control) **any → (integer / real / string)**
The content of Tuple is stored by Vision Q.400 to the file. It is intended to contain information which does not belong to the output_object Output, but can be used in another way, too.

2.12 OnCreateInstance

Name

OnCreateInstance — called when a checker instance is created.

Synopsis

OnCreateInstance(: : Cookie : NewCookie)

Description

OnCreateInstance is called when a new instance of the user defined checker is created, and inserted into the sequence list.

Parameters

Cookie (input_control) **integer → (integer)**
Cookie which may help to identify a user defined checker instance. The cookie may especially be useful if the script manages different information for each instance.

NewCookie (output_control) **integer → (integer)**
If set, this new Cookie value will be assigned to this checker instance.

2.13 OnCopyInstance

Name

OnCopyInstance — called when a checker instance is copied.

Synopsis

OnCopyInstance(ObjectToTransfer : NewObjectToTransfer : Cookie, CookieSource, TupleToTranfer : NewTupleToTransfer, NewCookie)

Description

OnCopyInstance is called when a new instance of the user defined checker is copied, and inserted into the sequence list.

Parameters

ObjectToTransfer (input_object) **object(-array)→ object**
The transfer object of the checker instance. It is a copy of the transfer object of the checker instance the current checker instance is a copy of.

NewObjectToTransfer (output_object) **object(-array) → object**
The new transfer object of the checker instance. If ObjectToTransfer is updated by the checker instance the result of the update has to be returned in New ObjectToTransfer.

Cookie (input_control) **integer → (integer)**
Cookie which may help to identify a user defined checker instance. The cookie may especially be useful if the script manages different information for each instance.

CookieSource (input_control) **integer → (integer)**
Cookie of the checker that was copied. The cookie may especially be useful if the script manages different information for each instance. It may be necessary to copy information from the source checker.

TupleToTransfer (input_control) **any → (integer / real / string)**
The transfer tuple of the checker instance. It is a copy of the transfer tuple of the checker instance the current checker instance is a copy of.

NewCookie (output_control) **integer → (integer)**
If set, this new Cookie value will be assigned to this checker instance.

New TupleToTransfer (output_control) **any → (integer / real / string)**
The new transfer tuple of the checker instance. If TupleToTransfer is updated by the checker instance the result of the update has to be returned in NewTupleToTransfer.

2.14 OnDeleteInstance

Name

OnDeleteInstance — called when a checker instance is deleted.

Synopsis

OnDeleteInstance(: : Cookie)

Description

OnDeleteInstance is called when an instance of the user defined checker is deleted.

Parameters

Cookie (input_control) **integer → (integer)**

Cookie which may help to identify a user defined checker instance. The cookie may especially be useful if the script manages different information for each instance.

2.15 OnExecuteInstance

Name

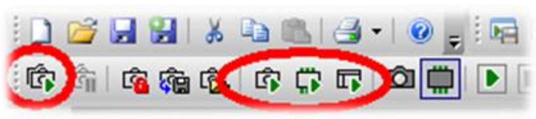
OnExecuteInstance — called when a checker instance is executed.

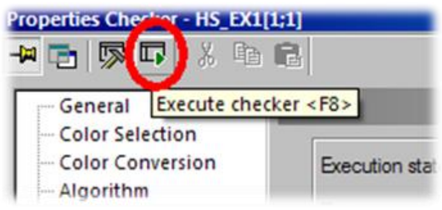
Synopsis

OnExecuteInstance(Image, Shape, GrayResults, BinaryResults, ObjectToTransfer : NewImage, ObjectsWithResults, ObjectsToDraw, ImageToDraw, ContoursToDraw, NewObjectToTransfer : Cookie, ParameterValues, UsedResultValues, ImageSource, AdjustedShape, TupleToTransfer : DetectedObjects, Results, ErrorMessage, EmptyContourValue, NewParameterValues, NewTupleToTransfer, ExecuteInstanceResult, ColorsForObjectsWithResults, ColorsForObjectsToDraw, ObjectsDrawMode)

Description

OnExecuteInstance is called when an instance of the user defined checker is executed.





Parameters

Image (input_object)	**image → object**
The current image. If the checker does not use gray or binary image filters, the domain of the image is the full image. If it does use gray or binary image filters, the domain of the image is reduced to the current shape.	
Shape (input_object)	**region → object**
The current shape. This means if the checker uses a position/rotation adjustment, the shape is adjusted according to this position/rotation adjustment.	
GrayResults (input_object)	**image → object**
Contains the results of this gray image filtering. The results are only calculated in the checker's current shape. If grey image filters are not used, this image will be empty. The pixels outside the shape of the checker are black.	
BinaryResults (input_object)	**region(-array) → object**
Contains the objects created by the thresholding and/or the binary image filters. If thresholding or binary image filters are not used, this region array will be empty.	
UsedResultObjects (input_object)	**object(-array)→ object**
If the procedure GetUsedResultDescriptions has defined some Used Results, the objects from which the results are calculated will be provided here. Otherwise the object (-array) will be empty. (See parameter UsedResultValues, too.) The index of an object in UsedResultObjects is the same as the index of the result value in UsedResultValues. That means that for every result UsedResultObjects contains to objects: an empty object at the position of the result type in UsedResultValues, which has to be ignored, and the object, from which the result is calculated, at the position of the result value in UsedResultValues.	
ObjectToTransfer (input_object)	**object(-array)→ object**
The transfer object of the checker instance.	
NewImage (output_object)	**image → object**
A new image. Note: this will be the new image for the whole application, and not just the checker instance. The new image must be the same size as the original image.	
ObjectsWithResults (output_object)	**region → object**
The objects for which results are calculated. If the checker does not calculate objects, but only returns results, you have to insert so many empty objects that the condition described under Results is fulfilled, that	

means that for each set of results one empty object has to be inserted.	
ObjectsToDraw (output_object)	**region → object**
Additional objects to be drawn with the ObjectsWithResults are placed in this region array. Corresponding objects in the arrays ObjectsWithResults and ObjectsToDraw share the same array position. How this array is treated depends on the setting of the parameter ObjectsDrawMode:	
ImageToDraw (output_object)	**image → object**
If the domain of ImageToDraw is not empty after the call of OnExecuteInstance, the image returned in ImageToDraw is used as processing value results of the execution. The domain of the returned image is not checked. That means if the domain of the returned image is not equal to the domain given by Shape, the returned image parts outside the shape are drawn, too. If ImageToDraw is derived either from Image or NewImage, it needs to refer to a different HALCON image object to that of Image or NewImage. It will be ignored if it refers to the same HALCON image object. ImageToDraw will refer to the same HALCON image object if it is created with the HALCON operator copy_obj. Therefore, to ensure that it refers to a different HALCON image object, use the operator copy_image. (For details, see the HALCON documentation regarding operators copy_obj and copy_image). NOTE: In order to see the ImageToDraw, the grey image processing results visibility must be turned ON. See this setting on the **Visibility** page of the checker properties dialog. [✓] Gray image processing results	
ContoursToDraw (output_object)	**xld_cont(-array) → object**
A XLD contour array that contains the contours that should be drawn. Contours to be drawn with the ObjectsWithResults are placed in this XLD contour array. Corresponding objects in the arrays ObjectsWithResults and ContoursToDraw share the same array position. How this array is treated depends on whether ObjectsWithResults is empty or not: ObjectsWithResults contains objects: This array can contain any number of objects, but the maximum number of objects drawn is the number of objects in the array ObjectsWithResults. ObjectsWithResults is empty: All objects in ContoursToDraw will be drawn. This is useful for checkers that have an OK state with no results, for example a checker that calculates differences.	
NewObjectToTransfer (output_object)	**object(-array) → object**
The new transfer object of the checker instance. If ObjectToTransfer is updated by the checker instance the result of the update has to be returned in New ObjectToTransfer.	
Cookie (input_control)	**integer → (integer)**
Cookie which may help to identify a user defined checker instance. The cookie may especially be useful if the script manages different information for each instance.	
ParameterValues (input_control)	**any → (integer / real / string)**
If parameters have been defined by the procedure GetParameterDescriptions, the current values will be provided here, in the same order that they were defined. Otherwise the parameter will be empty.	
UsedResultValues (input_control)	**any → (integer / real / string)**
If the procedure GetUsedResultDescriptions has defined some Used Results, their values will be provided here. Otherwise the parameter will be empty. There are two entries for each result, the first entry identifying the type of the second entry. The possible identifiers and their meaning is as follows.	

Identifier and Value of first entry.	**Meaning of second entry.**
DATA_INVALID = 0	The result value is given, but its value could not be calculated.

DATA_EMPTY = 1	The result value is an empty string (for string type results) or 0.0 (for real type results), or for a "Group" the value is the number of entries.
DATA_DOUBLE = 3	The result value is a double value.
DATA_STRING = 7	The result value is a string value.

ImageSource (input_control) — **any → (integer)**

Indicates the selected image source. The possible identifiers and their meaning are as follows.

Value	**Meaning.**
IMAGE_GRAY = 0	The checker does not use an image source. This is especially true if the checker works on a gray value image.
IMAGE_COLOR = 1	Currently selected image source is the color image.
IMAGE_RED_CH = 258 (0x102)	Currently selected image source is the red channel of the color image.
IMAGE_GREEN_CH = 514 (0x202)	Currently selected image source is the green channel of the color image.
IMAGE_BLUE_CH = 770 (0x302)	Currently selected image source is the blue channel of the color image.
IMAGE_HUE_CH = 1026 (0x402)	Currently selected image source is the hls hue channel of the color image.
IMAGE_LUM_CH = 1282 (0x502)	Currently selected image source is the hls luminance channel of the color image.
IMAGE_SAT_CH = 1538 (0x602)	Currently selected image source is the hls saturation channel of the color image.
IMAGE_HSV_HUE_CH = 1794 (0x702)	Currently selected image source is the hsv hue channel of the color image.
IMAGE_HSV_SAT_CH = 2050 (0x802)	Currently selected image source is the hsv saturation channel of the color image.
IMAGE_HSV_VAL_CH = 2306 (0x902)	Currently selected image source is the hsv value channel of the color image.
IMAGE_COM_RGB_CH = 2562 (0xA02)	Currently selected image source is a gray value combination of the three rgb channels of the color image.

AdjustedShape (input_control) — **any → (integer / real)**

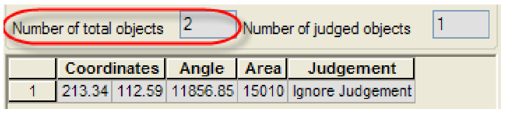
Information about the current state of the checker's shape. They contain the shape type, the current values of the position rotation adjusted of the checker, and the not adjusted shape information as shown in the checker's shape property page. For details see Adjusted Shape Information.

TupleToTransfer (input_control) — **any → (integer / real / string)**

The transfer tuple of the checker instance.

DetectedObjects (output_control) — **any → (integer)**

Number of detected objects. This number needs to be bigger than or equal to the number of objects, for which the Results are calculated.

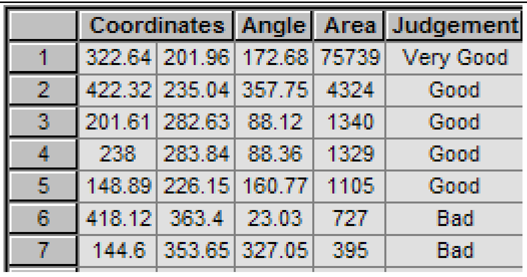


Number of total objects: 2 — Number of judged objects: 1

	Coordinates		Angle	Area	Judgement
1	213.34	112.59	11856.85	15010	Ignore Judgement

Results (output_control) — **any → (integer / real / string)**

For each detected object there is a set of results. The set is defined by the GetResultDescriptions procedure.
Example: If GetResultsDescriptions defines 5 result values (see example in GetResultsDescriptions above) and the number of result objects is 4, then the output tuple Results must contain 4 x 5 = 20 entries.
The tuple array first contains the five 'X' Coordinate results, followed by the five 'Y' Coordinate results, and so on for the other results.
The results would then be displayed in the Results tab of the checkers' property page. e.g.



	Coordinates		Angle	Area	Judgement
1	322.64	201.96	172.68	75739	Very Good
2	422.32	235.04	357.75	4324	Good
3	201.61	282.63	88.12	1340	Good
4	238	283.84	88.36	1329	Good
5	148.89	226.15	160.77	1105	Good
6	418.12	363.4	23.03	727	Bad
7	144.6	353.65	327.05	395	Bad

ErrorMessage (output_control) **string → (string)**

A message that is displayed when ExecuteInstanceResult is not 0

EmptyContourValue (output_control) **any → (integer/real)**

A Control output variable that can be set to the value that an empty contour is specified as. The default is the point -1,-1. If the point -1,-1 happens to be a valid contour in the set of ContoursToDraw, you can define a different value here.

NewParameterValues (output_control) **any → (integer / real / string)**

An output tuple that describes new values for parameters described by the procedure GetParameterDescriptions. This is particularly useful when hidden parameters are used, and the script needs to change the values.
Each new parameter value is described by three tuple entries as follows:

Entry No.	Type	Description
0	string	Name of group to which the parameter belongs.
1	string	Name of parameter.
2	integer/real/string (according to how the parameter was defined in GetParameterDescriptions)	New value of parameter.

NewTupleToTransfer (output_control) **any → (integer / real / string)**

The new transfer tuple of the checker instance. If TupleToTransfer is updated by the checker instance the result of the update has to be returned in NewTupleToTransfer.

ExecuteInstanceResult (output_control) **integer → (integer)**

Result of the OnExecuteInstance procedure.

Possible values are:
0 => The execution of the user defined checker instance was OK. (Default)
1 => The execution of the user defined checker instance produced a warning.
2 => The execution of the user defined checker instance produced an error.

ColorsForObjectsWithResults (output_control) **integer → (integer)**

In this tuple the color, which is used in the Vision Q.400 camera window, for every object with results may be returned: the first color in the tuple is assigned to the first object with results, the second to the second, and so on. If the tuple contains less colors than found objects with results, the colors are used "turn around". If it contains more colors, the unnecessary colors are ignored.

If the tuple is not returned, for all objects the color set in Vision Q.400 is used. The same is true if the empty tuple is returned.

ColorsForObjectsToDraw (output_control) **integer → (integer)**

In this tuple the color, which is used in the Vision Q.400 camera window, for every object to draw may be returned: the first color in the tuple is assigned to the first object with results, the second to the second, and so on. If the tuple contains less colors than found objects with results, the colors are used "turn around". If it contains more colors, the unnecessary colors are ignored.

If the tuple is not returned, for all objects the color set in Vision Q.400 is used. The same is true if the empty

tuple is returned.	
ObjectsDrawMode (output_control)	**integer → (integer)**
In this tuple the drawing mode is adjusted: Possible values are: 0 => The display of the ObjectsToDraw is related to the content of the Objects with results ObjectsWithResults contains objects: This array can contain any number of objects, but the maximum number of objects drawn is the number of objects in the array ObjectsWithResults. ObjectsWithResults is empty: All objects in ObjectsToDraw will be drawn. This is useful for checkers that have an OK state with no results, for example a checker that calculates differences. 1 => The display of the ObjectsToDraw is done totally independent The ObjectsToDraw array can contain any number of objects. All objects are drawn in the dedicated color. If the tuple is not returned, the ObjectsDrawMode is set to '0'. The same is done if the empty tuple is returned.	

2.15.1 Adjusted Shape Information

2.15.1.1 Rectangle, Rectangle->

Exampl e	AdjustedShape	=	[16, 0, 0, 0.0, 163, 295, 145, 246, 182, 345, 270.0]			
Length	|AdjustedShape|	=	11			
Values	AdjustedShape[0]	=	Shape type	0, 16	long	0 for rectangle 16 for rectangle ->
	AdjustedShape[1]	=	Pos. Rot. Adj.	Delta X	double	
	AdjustedShape[2]	=		Delta Y	double	
	AdjustedShape[3]	=		Delta Angle	double	
	AdjustedShape[4]	=	Center Point	X	long	
	AdjustedShape[5]	=		Y	long	
	AdjustedShape[6]	=	Start Point	X	long	
	AdjustedShape[7]	=		Y	long	
	AdjustedShape[8]	=	End Point	X	long	
	AdjustedShape[9]	=		Y	long	
	AdjustedShape[10]	=	Direction Angle	Angle	double	

2.15.1.2 Ellipse, Circle

Example	AdjustedShape	=	[1, 0, 0, 0.0, 340, 291, 201, 193, 479, 389]			
Length	|AdjustedShape|	=	10			
Values	AdjustedShape[0]	=	Shape type	1, 131073	long	1 for ellipse, 131073 for circle

	AdjustedShape[1]	=	Pos. Rot. Adj.	Delta X	double	
	AdjustedShape[2]	=		Delta Y	double	
	AdjustedShape[3]	=		Delta Angle	double	
	AdjustedShape[4]	=	Center Point	X	long	
	AdjustedShape[5]	=		Y	long	
	AdjustedShape[6]	=	Top Left Point	X	long	
	AdjustedShape[7]	=		Y	long	
	AdjustedShape[8]	=	Bottom Right Point	X	long	
	AdjustedShape[9]	=		Y	long	

2.15.1.3 Ellipse->, Circle->

Example	AdjustedShape	=	[17, 14, -68, -21.48, 294, 176, 222, 104, 366, 248, 360, 148, 245, 229, 0]			
Length	|AdjustedShape|	=	15			
Values	AdjustedShape[0]	=	Shape type	17, 131089	long	17 for ellipse ->, 131089 for circle->
	AdjustedShape[1]	=	Pos. Rot. Adj.	Delta X	double	
	AdjustedShape[2]	=		Delta Y	double	
	AdjustedShape[3]	=		Delta Angle	double	
	AdjustedShape[4]	=	Center Point	X	long	
	AdjustedShape[5]	=		Y	long	
	AdjustedShape[6]	=	Top Left Point	X	long	
	AdjustedShape[7]	=		Y	long	
	AdjustedShape[8]	=	Bottom Right Point	X	long	
	AdjustedShape[9]	=		Y	long	
	AdjustedShape[10]	=	Start Point	X	long	
	AdjustedShape[11]	=		Y	long	
	AdjustedShape[12]	=	End Point	X	long	
	AdjustedShape[13]	=		Y	long	
	AdjustedShape[14]	=	Scanning Direction	0	long	always 0, use start and end point

2.15.1.4 Doughnut

Example	AdjustedShape	=	[7, 0, 0, 0.0, 154, 239, 52, 137, 107, 192, 256, 235, 116, 144]			
Length	|AdjustedShape|	=	14			
Values	AdjustedShape[0]	=	Shape type	7	long	7 for doughnut

	AdjustedShape[1]	=	Pos. Rot. Adj.	Delta X	double	
	AdjustedShape[2]	=		Delta Y	double	
	AdjustedShape[3]	=		Delta Angle	double	
	AdjustedShape[4]	=	Center Point	X	long	
	AdjustedShape[5]	=		Y	long	
	AdjustedShape[6]	=	Outer circle edge point	X	long	
	AdjustedShape[7]	=		Y	long	
	AdjustedShape[8]	=	Inner circle edge point	X	long	
	AdjustedShape[9]	=		Y	long	
	AdjustedShape[10]	=	Start Point	X	long	
	AdjustedShape[11]	=		Y	long	
	AdjustedShape[12]	=	End Point	X	long	
	AdjustedShape[13]	=		Y	long	

2.15.1.5 Doughnut->

Exampl e	AdjustedShape	=	[19, 0, 0, 0.0, 81, 160, 44, 123, 58, 137, 49, 142, 45, 167, 1]			
Length	|AdjustedShape|	=	15			
Values	AdjustedShape[0]	=	Shape type	19	long	19 for doughnut ->
	AdjustedShape[1]	=	Pos. Rot. Adj.	Delta X	double	
	AdjustedShape[2]	=		Delta Y	double	
	AdjustedShape[3]	=		Delta Angle	double	
	AdjustedShape[4]	=	Center Point	X	long	
	AdjustedShape[5]	=		Y	long	
	AdjustedShape[6]	=	Outer circle edge point	X	long	
	AdjustedShape[7]	=		Y	long	
	AdjustedShape[8]	=	Inner circle edge point	X	long	
	AdjustedShape[9]	=		Y	long	
	AdjustedShape[10]	=	Start Point	X	long	
	AdjustedShape[11]	=		Y	long	
	AdjustedShape[12]	=	End Point	X	long	
	AdjustedShape[13]	=		Y	long	
	AdjustedShape[14]	=	Scanning Direction	0, 1	long	0 = clockwise, 1 = counterclockwise

2.15.1.6 Line->

Exampl e	AdjustedShape	=	[18, 0, 0, 0.0, 217, 275, 95, 348, 340, 203, 30.6186]			
Length	|AdjustedShape|	=	11			

Values	AdjustedShape[0]	=	Shape type	18, 65554, 131090	long	18 for line->, 65554 for horizontal line->, 131090 for vertical line->
	AdjustedShape[1]	=	Pos. Rot. Adj.	Delta X	double	
	AdjustedShape[2]	=		Delta Y	double	
	AdjustedShape[3]	=		Delta Angle	double	
	AdjustedShape[4]	=	Center Point	X	long	
	AdjustedShape[5]	=		Y	long	
	AdjustedShape[6]	=	Start Point	X	long	
	AdjustedShape[7]	=		Y	long	
	AdjustedShape[8]	=	End Point	X	long	
	AdjustedShape[9]	=		Y	long	
	AdjustedShape[10]	=	Angle		double	

2.15.1.7 Polygon

Example	AdjustedShape	=	[3, 0, 0, 0.0, 286, 202, 192, 225, 243, 140, 362, 146, 380, 191, 297, 265]			
Length	|AdjustedShape|	=	6 + 2 * no of polygon points			
Values	AdjustedShape[0]	=	Shape type	3	long	3 for polygon
	AdjustedShape[1]	=	Pos. Rot. Adj.	Delta X	double	
	AdjustedShape[2]	=		Delta Y	double	
	AdjustedShape[3]	=		Delta Angle	double	
	AdjustedShape[4]	=	Center Point	X	long	
	AdjustedShape[5]	=		Y	long	
	AdjustedShape[6]	=	1st Point	X	long	
	AdjustedShape[7]	=		Y	long	
	..		..	..	..	
	AdjustedShape[2n+4]	=	n-th Point	X	long	
	AdjustedShape[2n+5]	=		Y	long	

2.15.1.8 Dynamic Object Shape

Example	AdjustedShape	=	[10]			
Length	|AdjustedShape|	=	1			
Value	|AdjustedShape|	=	Shape type	3	long	10 for dynamic object shape

2.15.1.9 Static Object Shape

Exampl e	AdjustedShape	=	[65546, 0, 0, 0.0]			
Length	|AdjustedShape|	=	4			
Values	AdjustedShape[0]	=	Shape type	65546	long	65546 for static object shape
	AdjustedShape[1]	=	Pos. Rot. Adj.	Delta X	double	
	AdjustedShape[2]	=		Delta Y	double	
	AdjustedShape[3]	=		Delta Angle	double	

2.15.1.10 No Shape

Example	AdjustedShape	=	
Length	|AdjustedShape|	=	0

2.16 OnButtonPressed

Name

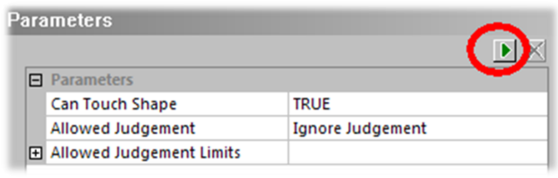
OnButtonPressed — called when the button on the Parameters or Used Results page is pressed.

Synopsis

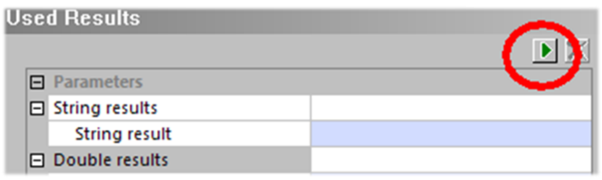
OnButtonPressed(Image, Shape, GrayResults, BinaryResults, ObjectToTransfer : NewImage, ObjectsToDraw, ImageToDraw, ContoursToDraw, NewObjectToGransfer : Cookie, ParameterButtonPressed, InstanceInfoParameters, ParameterValues, InstanceInfoUsedResults, UsedResultValues, ImageSource, AdjustedShape, TupleToTransfer : EmptyContourValue, NewParameterValues, NewTupleToTransfer, ButtonPressedResult)

Description

OnButtonPressed is called whenever the button on the Parameters or Used Results page is clicked.


Parameters

⊟ Parameters	
Can Touch Shape	TRUE
Allowed Judgement	Ignore Judgement
⊞ Allowed Judgement Limits	


Used Results

⊟ Parameters	
⊟ String results	
String result	
⊟ Double results	

Parameters

Image (input_object) **image → object**
The current image. If the checker does not use gray or binary image filters, the domain of the image is the full image. If it does use gray or binary image filters, the domain of the image is reduced to the current shape.

Shape (input_object) **region → object**
The current shape. This means if the checker uses a position/rotation adjustment, the shape is adjusted according to this position/rotation adjustment.

GrayResults (input_object) **image → object**
Contains the results of this gray image filtering. The results are only calculated in the checker's current shape. If grey image filters are not used, this image will be empty. The pixels outside the shape of the checker are black.

BinaryResults (input_object) **region(-array) → object**
Contains the objects created by the thresholding and/or the binary image filters.

UsedResultObjects (input_object) **object(-array)→ object**
If the procedure GetUsedResultDescriptions has defined some Used Results, the objects from which the results are calculated will be provided here. Otherwise the object (-array) will be empty. (See parameter UsedResultValues, too.)

The index of an object in UsedResultObjects is the same as the index of the result value in UsedResultValues. That means that for every result UsedResultObjects contains to objects: an empty object at the position of the result type in UsedResultValues, which has to be ignored, and the object, from which the result is calculated, at the position of the result value in UsedResultValues.

ObjectToTransfer (input_object) **object(-array)→ object**
The transfer object of the checker instance.

NewImage (output_object) **image → object**
A new image. Note: this will be the new image for the whole application, and not just the checker instance. The new image must be the same size as the original image.

ObjectsToDraw (output_object) **region(-array) → object**
Objects that are drawn if the ButtonPressedResult value is set to 2.

ImageToDraw (output_object) **image → object**
Image that is drawn if the ButtonPressedResult value is set to 2.
The domain of image is not checked. That means if the domain of the image extends beyond the domain of the Shape, the image outside the shape is also drawn.

If ImageToDraw is derived either from Image or NewImage, it needs to refer to a different HALCON image object to that of Image or NewImage. It will be ignored if it refers to the same HALCON image object. ImageToDraw will refer to the same HALCON image object if it is created with the HALCON operator copy_obj. Therefore, to ensure that it refers to a different HALCON image object, use the operator copy_image. (For details, see the HALCON documentation regarding operators copy_obj and copy_image).

ContoursToDraw (output_object) **xld_cont(-array) → object**

A XLD contour array that contains the contours that should be drawn if the ButtonPressedResult value is set to 2.

If contours in ContoursToDraw are drawn, the objects in ObjectsToDraw are not drawn.

NewObjectToTransfer (output_object) **object(-array) → object**

The new transfer object of the checker instance. If ObjectToTransfer is updated by the checker instance the result of the update has to be returned in New ObjectToTransfer.

Cookie (input_control) **integer → (integer)**

Cookie which may help to identify a user defined checker instance. The cookie may especially be useful if the script manages different information for each instance.

ParameterButtonPressed (input_control) **integer → (integer)**

Informs which button has been pressed (See Description)

1 => Execute Parameters

0 => Execute Used Results

InstanceInfoParameters (input_control) **any → (integer)**

Informs about the selected parameter group and the selected parameter when OnButtonPressed is called. It can be used to implement a step by step testing of the selected parameter values.

This is a pair of tuples, the first being the index of the currently selected group (-1 if no group is selected) and the second being the index of the currently selected instance in the group (-1 if no instance is selected).

ParameterValues (input_control) **any → (integer / real / string)**

If parameters have been defined by the procedure **GetParameterDescriptions**, the current values will be provided here, in the same order that they were defined. Otherwise the parameter will be empty.

InstanceInfoUsedResults (input_control) **any → (integer)**

Informs about the selected used results group and the selected used result when OnButtonPressed is called. It can be used to implement a step by step testing of the selected parameter values.

This is a pair of tuples, the first being the index of the currently selected group (-1 if no group is selected) and the second being the index of the currently selected instance in the group (-1 if no instance is selected).

The tuple is empty if the "Used Results" are not used, or if the ParameterButtonPressed is 1 (Execute Parameters).

UsedResultValues (input_control) **any → (integer / real / string)**

If parameters have been defined by the procedure **GetUsedResultDescriptions**, the current Used Result values will be provided here, in the same order that they were defined. Otherwise the parameter will be empty.

ImageSource (input_control) **any → (integer)**

Indicates the selected image source. The possible identifiers and their meaning are as follows.

Value	**Meaning.**
IMAGE_GRAY = 0	The checker does not use an image source. This is especially true if the checker works on a gray value image.
IMAGE_COLOR = 1	Currently selected image source is the color image.
IMAGE_RED_CH = 258 (0x102)	Currently selected image source is the red channel of the color image.
IMAGE_GREEN_CH = 514 (0x202)	Currently selected image source is the green channel of the color image.
IMAGE_BLUE_CH = 770 (0x302)	Currently selected image source is the blue channel of the color image.
IMAGE_HUE_CH = 1026 (0x402)	Currently selected image source is the hls hue channel of the color image.
IMAGE_LUM_CH = 1282 (0x502)	Currently selected image source is the hls luminance channel of the color image.

IMAGE_SAT_CH = 1538 (0x602)	Currently selected image source is the hls saturation channel of the color image.
IMAGE_HSV_HUE_CH = 1794 (0x702)	Currently selected image source is the hsv hue channel of the color image.
IMAGE_HSV_SAT_CH = 2050 (0x802)	Currently selected image source is the hsv saturation channel of the color image.
IMAGE_HSV_VAL_CH = 2306 (0x902)	Currently selected image source is the hsv value channel of the color image.
IMAGE_COM_RGB_CH = 2562 (0xA02)	Currently selected image source is a gray value combination of the three rgb channels of the color image.

AdjustedShape(input_control) **any → (integer / real)**

Information of the current state of the checker's shape. They contain the shape type, the current values of the position rotation adjusted of the checker, and the not adjusted shape information as shown in the checker's shape property page. For details see Adjusted Shape Information.

TupleToTransfer (input_control) **any → (integer / real / string)**

The transfer tuple of the checker instance.

EmptyContourValue (output_control) **any → (integer/real)**

A Control output variable that can be set to the value that an empty contour is specified as. The default is the point -1,-1. If the point -1,-1 happens to be a valid contour in the set of ContoursToDraw, you can define a different value here.

NewParameterValues (output_control) **any → (integer / real / string)**

An output tuple that describes new values for parameters described by the procedure GetParameterDescriptions. This is particularly useful when hidden parameters are used, and the script needs to change the values.

Each new parameter value is described by three tuple entries as follows:

Entry No.	Type	Description
0	string	Name of group to which the parameter belongs.
1	string	Name of parameter.
2	integer/real/string (according to how the parameter was defined in GetParameterDescriptions)	New value of parameter.

NewTupleToTransfer (output_control) **any → (integer / real / string)**

The new transfer tuple of the checker instance. If TupleToTransfer is updated by the checker instance the result of the update has to be returned in NewTupleToTransfer.

ButtonPressedResult (output_control) **integer → (integer)**

Result of the OnButtonPressed procedure. Possible values are:

0 => Draw nothing.

1 => Draw grey image filtering results along with ObjectsToDraw

2 => Draw the image in ImageToDraw and the objects in ObjectsToDraw if they are given.

3 Debugging Support

One big advantage of scripting is the ability to leverage the power of the HDevelop editor and debugger. In order to easily debug scripts written for Vision Q.400, there are external procedures provided. These will write debug data (image, parameter values, region information) during execution within Vision Q.400 to hard disk, and then read the data when executing within HDevelop.
This ensures that you can run and debug a script in HDevelop with exactly the same data as provided by Vision Q.400.
For comfortable handling we recommend implementing a script parameter to enable and disable this feature from within Vision Q.400 (see screenshot below).

Debug Procedures

This group of procedures enhances the debugging possibilities of scripts running within Vision Q.400. The procedures are external to the checker script and can be found in the sub-folder **.\User Defined Checkers\HDevelop\External Procedures** of the Vision Q.400 installation folder.

Procedure name	Description
Test_InHDevelop	Check if the script is running within Vision Q.400 or within HDevelop.
ReadDebugData_OnExecuteInstance	Read the debug data when run by HDevelop from the folder **C:\Debug**
WriteDebugData_OnExecuteInstance	Write the debug data when run by Vision Q.400 into the folder **C:\Debug**

To use the debug procedures you need to add the following lines of code in the procedure **OnExecuteInstance** before the parameter values are read.

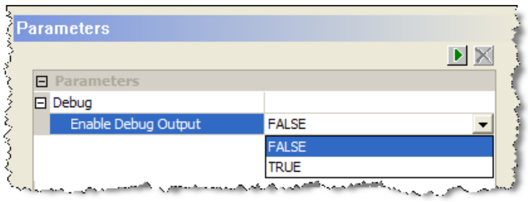
To control debug output, just comment in or out **WriteDebugData_OnExecuteInstance**.

```
*************************************************************************************************
*** For Debugging
*************************************************************************************************
try
* We determine if the script is running in HDevelop
TEST_InHDevelop (bInHDevelop)

if (bInHDevelop)
* Running in HDevelop, Read the debug information
ReadDebugData_OnExecuteInstance (Image, Shape, GrayResults, BinaryResults, Cookie, ParameterValues, UsedResultValues)
else
* Running in Vision Q.400, Write the debug information.
WriteDebugData_OnExecuteInstance (Image, Shape, GrayResults, BinaryResults, Cookie, ParameterValues, UsedResultValues)
endif

catch(E)
ExecuteInstanceResult := 2
ErrorMessage := 'Reading or Writing Debug Information failed'
throw(E)
endtry
```

Alternatively, it is possible to use a checker parameter to control the debug output.
Note that this is a parameter setting for a particular checker instance in Vision Q.400, and not for all instances.



Enable Debug Output: FALSE
FALSE
TRUE

For example, the following code in **GetParameterDescriptions** defines a debug parameter.

```
* Define a debug parameter. Default value is ‘false’ => off.
ParameterDescriptions := [ParameterDescriptions, 'Group', 'Debug@A@Par_Debug', '']
ParameterDescriptions := [ParameterDescriptions, 'Bool', 'Enable Debug Output@A@Par_DebugOutput', 'false','Create debug
                              information.']
```

In **OnExecuteInstance**, the parameter can be read and used to conditionally execute the writing of debug information (in this example the parameter is located at index 18, but this will

```
* Read the parameter that determines whether debug information should be written.
Par_DebugOutput := ParameterValues[18]

* If the debug parameter is true, and we are NOT in HDevelop (i.e. in Vision Q.400), write the debug information
if (Par_DebugOutput = 'true' and bInHDevelop = 0)
WriteDebugData_OnExecuteInstance (Image, Shape, GrayResults, BinaryResults,Cookie, ParameterValues, UsedResultValues)
Endif
```

4 Script Helpers

The Vision Q.400 installation contains some HDevelop procedures you can use to facilitate the programming of your script.

Available Procedures

This group of procedures facilitates the programming of your script in different aspects related to the provided procedure. The procedures are external to the checker script and can be found in the sub-folder **.\User Defined Checkers\HDevelop\External Procedures** of the Vision Q.400 installation folder.

Procedure name	Description
Convert_RGB_COLORREF	Convert a tuple containing an RGB color value into a tuple containing a COLORREF value used in Windows. This is helpful to covert the color values used for drawing with Halcon inside the script into color values you can provide as parameter for displaying the ObjectsWithResults and the ObjectsToDraw in different colors. **Convert_RGB_COLORREF**(: : *ColorRGB* : *ColorCOLORREF*) Parameter **ColorRGB** (input_control) Tuple containg the color as RGB value in the format: [R,G,B] any → (*tuple*) **ColorCOLORREF** (output_control) Tuple containing the color in the COLORREF format used in Windows. any → (*tuple*)

5 Global Script Variables

Scripts can contain global variables. It should be noted however that these variables are global across all scripts used by Vision Q.400. This means that the variables come into existence when the scripts are loaded (when Vision Q.400 is started), and persist as long as Vision Q.400 is running. Therefore, if two scripts use a global variable of the same name, it could cause unexpected results. On the other hand, it may be very useful to be able to modify a single variable from multiple scripts.

If a script is developed for general distribution, it is not possible to know the global variable names used by other scripts. By pre or post pending the variable names with an identifier, like the checker's short name, a clash can be minimized.